Big data system analyzing Amazon reviews and reviewers

Introduction

We will spare the reader of any theoretical aspect usually presented in the introductory part, and we will instead present our between-the-lines take on the project, only to later go into details regarding each part. What we believe, must be mentioned from the get go, is that this task was more in the spectrum of big data analytics, rather than the whole concept of big data, which is associated more with managing data and IT infrastructure for serving it. This is the reason why we have been more focused on the idea of accurately predicting which review is fake and which is not. We have also taken the project a step further, rather than just analyzing and evaluating a single review/reviewer, we analyze whole products and modify their overall score based on the "fakes" we find.

Python was our so called "weapon" in this fight, and we have used a multitude of libraries for different targets, such as 'sqlite', 'flask', 'numpy', 'json', 'shutil', 'pandas', 'matplotlib', 'os', 'time', and 'math'. Everything was done through python, from the ingestion, to the processing and to the loading onto the website. Our choice for data storage was SQL, more specifically SQLite, which we will justify in the upcoming chapters.

In order to categorize modify the overall score, we have 2 layers of complexity regarding the issue, the "overall" layer, which takes into consideration all the reviews and all the reviewers, and the "individual" layer, which takes into consideration individual user behavior and the way it affects the grade.

Data Ingestion

Data ingestion is actually done through python, where we have unknowingly created an ETL pipeline, only to later learn about it, reading line after line from a

"json" file. The "json" file was actually the key factor in deciding not to use other ingestion methods, such as Kafka, or MQTT which have been presented in the course. Having a well-defined file (in the sense that all the data was written in different files on the github we have been provided with), gave us the assurance we needed regarding the flow of the incoming data,

which was not variable in any way, because, obviously, we were controlling how fast the data was coming in, and that was line by line.

The raw data had a lot of fields: Overall score, Verified Flag, Review Time, Reviewer ID, Product ID, Style, Reviewer Name, Review Text, Summary, unixReviewTime, out of which some were redundant and would have only occupied space. So this is were we go through the process of transformation (eTl) extracting the important information, such as a datetime type of date, after having modified the format so that it matches numpy's, the number of words each review has, a scan for incentivized reviews which results in a flag, the "bin" each review will be contained in(bins will be presented later), and the category the product belongs to. So we drop the style, the summary, the whole text, and the unixReviewTime, thus trimming our data.

The last part is loading (etL), which is simply done through the 'sqlite' library, allowing us to populate tables with the transformed data. Next we will explain why we have chosen SQL.

- # Data Storage

For storing data, we have FIRST chosen SQL, given the fact that we had to compute rather complex queries, which would have been much harder using NOSQL databases.(After the last labs, we have realized it would have been so much wiser to use Spark, but we had no idea at the beginning of the project, and transitioning would have been too complicated in this low interval of time). Another reason would be the clear structure of every file, which was repeated for every category of products. Therefore, we were assured that we always get the same type of input which we are able to insert into the database after transforming. And, in other to achieve full transparency, we choose SQL because we felt both of us were more fluent in this and would be able to achieve a better result. We have also indexed the Review table, so we have faster queries.

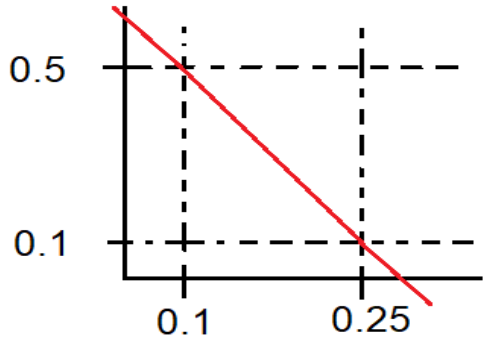
Why SQLite and not MySQL? Simply because the project has been treated as a toy example, where we wanted to test the results and wanted to achieve faster enquiries and access to the database, and we have done this through SQLite. Given the fact that the app only runs on our machines, this amplifies the speed. We do however know that for a larger app, with multiple users MySQL pays off in terms of writing (it does not hog the table by locking it when someone is writing inside), even though it has a slower connection.

- # Data Processing

We believe that this part is the bread and butter of our project. Previously, the idea of two layers of complexity has been brought up. We will first start with the "overall" layer.

The first criteria of analysis is based on the word count distribution. This has been one of the complex queries, given that it requires pivoting. The concept behind this is: we have a number of 10 bins, which contains the frequency of reviews with a certain number of words (0-15, 15-30, 30-45, …, 100+). We compare the overall category distribution(the category to which the product belongs), to the distribution created by the reviews of our product. And here , we split the problem in two parts:

a) A number of reviews higher than 25 (mathematical statistics teach us that a sampled quantity higher than 25 yields a distribution similar to the bigger quanity). In this case, we take into consideration two factors, the percentage of mis-distributed reviews, and the difference between the normally-distributed and mis-distributed. A review is counted as mis-distributed once it's frequency is 1.4 times higher than the category's. We do the final check using a linear function which has some user-defined values which it uses as limits for normality. The graph below explains it:



Anything beyond the red line is counted as fake. The red line is computed based on the limit points we were talking about.

b) The other case is the one were the number of reviews is lower than 25. In this situation, we use the big numbers law. We compute the category mean and standard deviation for the number of words, then we check how many of the reviews are outliers in terms of number of words (they're bigger or smaller than 2 times the standard dev). If the percentage of those is higher than 0.2, we simply return a warning, but we do not have enough data to accurately decipher the meaning of the warning.

The second criteria is the verified/unverified results. Here we compute how the unverified reviews affect the overall score. This procedure is much simpler, because unverified reviews have a much lower confidence "score" in our eyes, so if the absolute difference between the score with unverified reviews and the one without unverified reviews is higher than 0.1, we just delete all the unverified reviews. Another issue is having only unverified reviews, where we can obviously keep going with the algorithm, but this is a big enough question mark so that we just don't and return a big failure mark.

The third criteria is really similar to the second, it just analyses the incentivized/non-incentivized reviews. Amazon has a very strict policy which mentions that if a reviewer is incentivized, he must mention so in the review itself. Therefore, we check for the incentivized reviews, we do exactly the same operation as above and we eliminate the strange ones.

The fourth criteria is called rating trend. This one is tightly knit with an “individual” layer criteria, but we will explain it here nevertheless. It simply analyses the time when different users have reviewed the product, and if it so happens that multiple people have reviewed the product exactly in the same day – which would obviously be a distortion in the normal distribution – (just as above, we do check if a day is an outlier by comparing it to 2 times the standard deviation of the distribution) we mark the day as “suspicious”, and leave it for further investigation in the next phase.

Here we move on to the second layer of complexity, which is the “individual layer”. Here we see the impact each user’s behavior has on the final score.

The first criteria is called User Ease. This one takes into consideration how easy each user gives a grade, in the sense that a rather critique user who only gives ratings of 3 and 4 will have a higher say in the final grad than an “easy” user who always gives ratings of 5. So we just divide the “category ease” by the “user ease”, and that’s the coefficient we multiply the review’s rating with. Besides this, Amazon policy allows users to rate the same product multiple times if they purchase it multiple times, but this results in really biased reviews. Therefore, we do an average of each user’s score for the product and use it once, instead of using every value(we minimize the impact).

The second criteria is called User Overlapping History, and the concept behind it is based on the fact that users who have a higher overlapping history(compared to the average of each product), might be users paid by third parties who are also paid by companies to promote their products. The comparison criteria is just as above, outliers higher than twice the standard deviation.

The third and final criteria is called User Behaviour and is entwined with the Rating Trend. It is based on 4 flags:

-flag_post_day, which tells you how many reviews per day the user posted, we expect it to be around 1 or 2 per day, anything above that is classified as suspicious

-flag_number_verified, which tells you the amount of unverified reviews a user posts, which basically means he’s reviewing with no idea about the product

-flag_verified_rating, the difference a user has between average rating with and without verified reviews; if higher than 0.1 => suspicious

-more_revs_one_prod, tells you the user has a tendency of reviewing the same product multiple times and that his opinion tends to be biased.

If all 4 of these flags have a TRUE value, we classify the user as fake and just ignore the review/s completely. If 3 of them are true, we compare the days in which he has a suspicious activity (more than 3 posts per day), to the days in which the rating trend of the product has a weird activity (higher than mean + 2 std devs), and if the day in which the user reviewed the product is at the intersection of the 2 aforementioned categories, once again, we just normalize the rating of the review to the category norm.

- # Final Result

The final result is then uploaded on a locally-hosted website, where you can choose which product you want to analyse from which category. We would have attached some pictures, but we don't have enough space in the limit of 5 pages to do so, but we are more than willing to show the website in action during the oral presentation.

- # Conclusions

We want to start by saying that both of us can proudly say this project has been a lot of fun to work on, even more so given the fact that we had to come up with all these ideas by ourselves. We do know there is a lot of room for improvement when it comes to the infrastructure part, and we actually have a way of improving it by using redis, and we can present that if you'd like. Besides this, access to Amazon's API would have made the whole project a lot more interesting, but given we would have had to pay for it, we found it out of the scope of the task at hand. We wish to thank you for the opportunity to work on this, and are eager to wait your opinions and questions